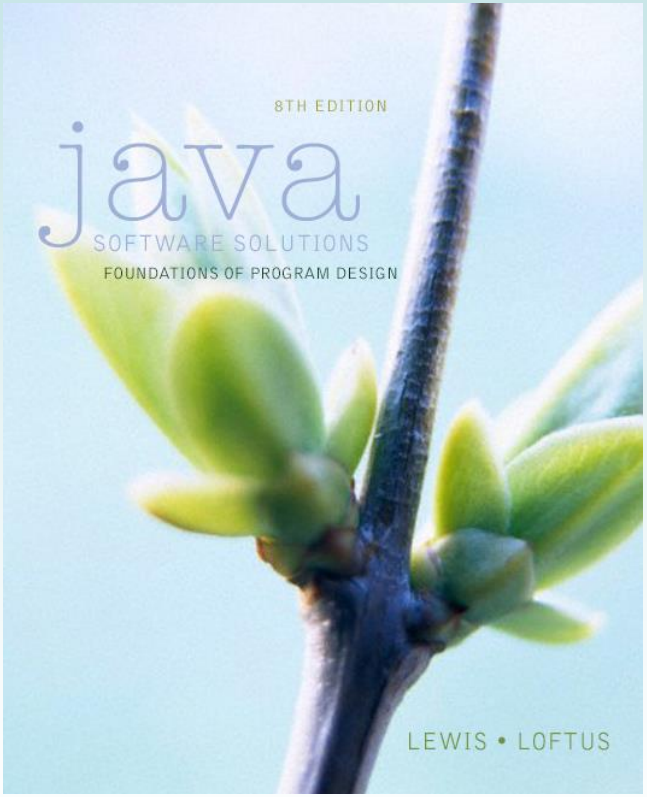
Chapter 4
Writing Classes



Java Software Solutions
Foundations of Program Design
8th Edition

John Lewis
William Loftus

Addison-Wesley
is an imprint of
PEARSON

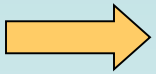
Copyright © 2014 Pearson Education, Inc.

Writing Classes

- We've been using predefined classes from the Java API. Now we will learn to write our own classes.
- Chapter 4 focuses on:
  - class definitions
  - instance data
  - encapsulation and Java modifiers
  - method declaration and parameter passing
  - constructors
  - graphical objects
  - events and listeners
  - buttons and text fields

Copyright © 2014 Pearson Education, Inc.

Outline



Anatomy of a Class

Encapsulation

Anatomy of a Method

Graphical Objects

Graphical User Interfaces

Buttons and Text Fields

Copyright © 2014 Pearson Education, Inc.

Writing Classes

- The programs we've written in previous examples have used classes defined in the Java standard class library
- Now we will begin to design programs that rely on classes that we write ourselves
- The class that contains the `main` method is just the starting point of a program
- True object-oriented programming is based on defining classes that represent objects with well-defined characteristics and functionality

Copyright © 2014 Pearson Education, Inc.

Examples of Classes

Class	Attributes	Operations
Student	Name Address Major Grade point average	Set address Set major Compute grade point average
Rectangle	Length Width Color	Set length Set width Set color
Aquarium	Material Length Width Height	Set material Set length Set width Set height Compute volume Compute filled weight
Flight	Airline Flight number Origin city Destination city Current status	Set airline Set flight number Determine status
Employee	Name Department Title Salary	Set department Set title Set salary Compute wages Compute bonus Compute taxes

Copyright © 2014 Pearson Education, Inc.

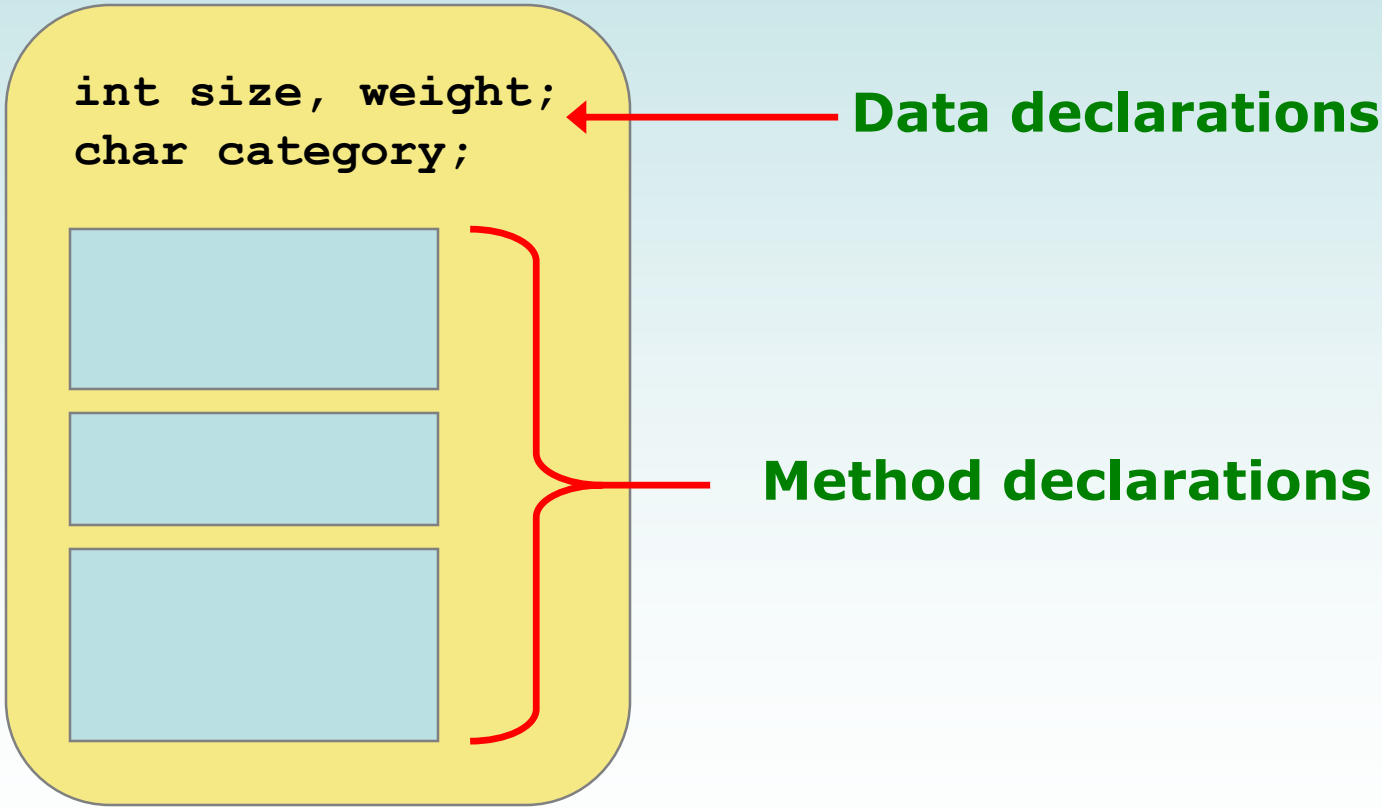
Classes and Objects

- Recall from our overview of objects in Chapter 1 that an object has *state* and *behavior*
- Consider a six-sided die (singular of dice)
  - It's state can be defined as which face is showing
  - It's primary behavior is that it can be rolled
- We represent a die by designing a class called `Die` that models this state and behavior
  - The class serves as the blueprint for a die object
- We can then instantiate as many die objects as we need for any particular program

Copyright © 2014 Pearson Education, Inc.

Classes

- A class can contain data declarations and method declarations



Copyright © 2014 Pearson Education, Inc.

Classes

- The values of the data define the state of an object created from the class
- The functionality of the methods define the behaviors of the object
- For our `Die` class, we might declare an integer called `faceValue` that represents the current value showing on the face
- One of the methods would “roll” the die by setting `faceValue` to a random number between one and six

Copyright © 2014 Pearson Education, Inc.

Classes

- We'll want to design the `Die` class so that it is a versatile and reusable resource
- Any given program will probably not use all operations of a given class
- See `RollingDice.java`
- See `Die.java`

Copyright © 2014 Pearson Education, Inc.

```
//********************************************************************
//  RollingDice.java       Author: Lewis/Loftus
//
//  Demonstrates the creation and use of a user-defined class.
//********************************************************************

public class RollingDice
{
   //-----------------------------------------------------------------
   //  Creates two Die objects and rolls them several times.
   //-----------------------------------------------------------------
   public static void main(String[] args)
   {
      Die die1, die2;
      int sum;

      die1 = new Die();
      die2 = new Die();

      die1.roll();
      die2.roll();
      System.out.println("Die One: " + die1 + ", Die Two: " + die2);
```

continue

Copyright © 2014 Pearson Education, Inc.

continue

```
        die1.roll();
        die2.setFaceValue(4);
        System.out.println("Die One: " + die1 + ", Die Two: " + die2);

        sum = die1.getFaceValue() + die2.getFaceValue();
        System.out.println("Sum: " + sum);

        sum = die1.roll() + die2.roll();
        System.out.println("Die One: " + die1 + ", Die Two: " + die2);
        System.out.println("New sum: " + sum);
    }
}
```

Copyright © 2014 Pearson Education, Inc.

continue

```
        die1.roll();
        die2.setFaceV
        System.out.pr                                    , Die Two: " + die2);

        sum = die1.ge                                    lue();
        System.out.pr

        sum = die1.roll() + die2.roll();
        System.out.println("Die One: " + die1 + ", Die Two: " + die2);
        System.out.println("New sum: " + sum);
    }
}
```

Sample Run

```
Die One: 5, Die Two: 2
Die One: 1, Die Two: 4
Sum: 5
Die One: 4, Die Two: 2
New sum: 6
```

Copyright © 2014 Pearson Education, Inc.

```
//********************************************************************
//  Die.java       Author: Lewis/Loftus
//
//  Represents one die (singular of dice) with faces showing values
//  between 1 and 6.
//********************************************************************

public class Die
{
   private final int MAX = 6;  // maximum face value

   private int faceValue;  // current value showing on the die

   //-----------------------------------------------------------------
   //  Constructor: Sets the initial face value.
   //-----------------------------------------------------------------
   public Die()
   {
      faceValue = 1;
   }
```

continue

Copyright © 2014 Pearson Education, Inc.

continue

```
//-----------------------------------------------------------------
//  Rolls the die and returns the result.
//-----------------------------------------------------------------
public int roll()
{
   faceValue = (int)(Math.random() * MAX) + 1;
   return faceValue;
}

//-----------------------------------------------------------------
//  Face value mutator.
//-----------------------------------------------------------------
public void setFaceValue(int value)
{
   faceValue = value;
}

//-----------------------------------------------------------------
//  Face value accessor.
//-----------------------------------------------------------------
public int getFaceValue()
{
   return faceValue;
}
```

continue

Copyright © 2014 Pearson Education, Inc.

continue

```
    //-----------------------------------------------------------------
    //  Returns a string representation of this die.
    //-----------------------------------------------------------------
    public String toString()
    {
        String result = Integer.toString(faceValue);

        return result;
    }
}
```

Copyright © 2014 Pearson Education, Inc.

The Die Class

- The `Die` class contains two data values
  - a constant `MAX` that represents the maximum face value
  - an integer `faceValue` that represents the current face value
- The `roll` method uses the `random` method of the `Math` class to determine a new face value
- There are also methods to explicitly set and retrieve the current face value at any time

Copyright © 2014 Pearson Education, Inc.

The toString Method

- It's good practice to define a `toString` method for a class
- The `toString` method returns a character string that represents the object in some way
- It is called automatically when an object is concatenated to a string or when it is passed to the `println` method
- It's also convenient for debugging problems

Copyright © 2014 Pearson Education, Inc.

Constructors

- As mentioned previously, a *constructor* is used to set up an object when it is initially created
- A constructor has the same name as the class
- The `Die` constructor is used to set the initial face value of each new die object to one
- We examine constructors in more detail later in this chapter

Copyright © 2014 Pearson Education, Inc.

Data Scope

- The *scope* of data is the area in a program in which that data can be referenced (used)
- Data declared at the class level can be referenced by all methods in that class
- Data declared within a method can be used only in that method
- Data declared within a method is called *local data*
- In the `Die` class, the variable `result` is declared inside the `toString` method -- it is local to that method and cannot be referenced anywhere else

Copyright © 2014 Pearson Education, Inc.

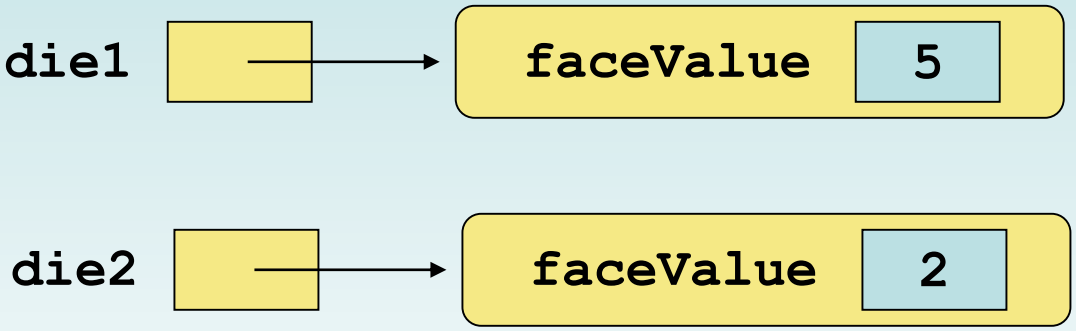
Instance Data

- A variable declared at the class level (such as `faceValue`) is called *instance data*
- Each instance (object) has its own instance variable
- A class declares the type of the data, but it does not reserve memory space for it
- Each time a `Die` object is created, a new `faceValue` variable is created as well
- The objects of a class share the method definitions, but each object has its own data space
- That's the only way two objects can have different states

Copyright © 2014 Pearson Education, Inc.

Instance Data

- We can depict the two `Die` objects from the `RollingDice` program as follows:



Each object maintains its own `faceValue` variable, and thus its own state

Copyright © 2014 Pearson Education, Inc.

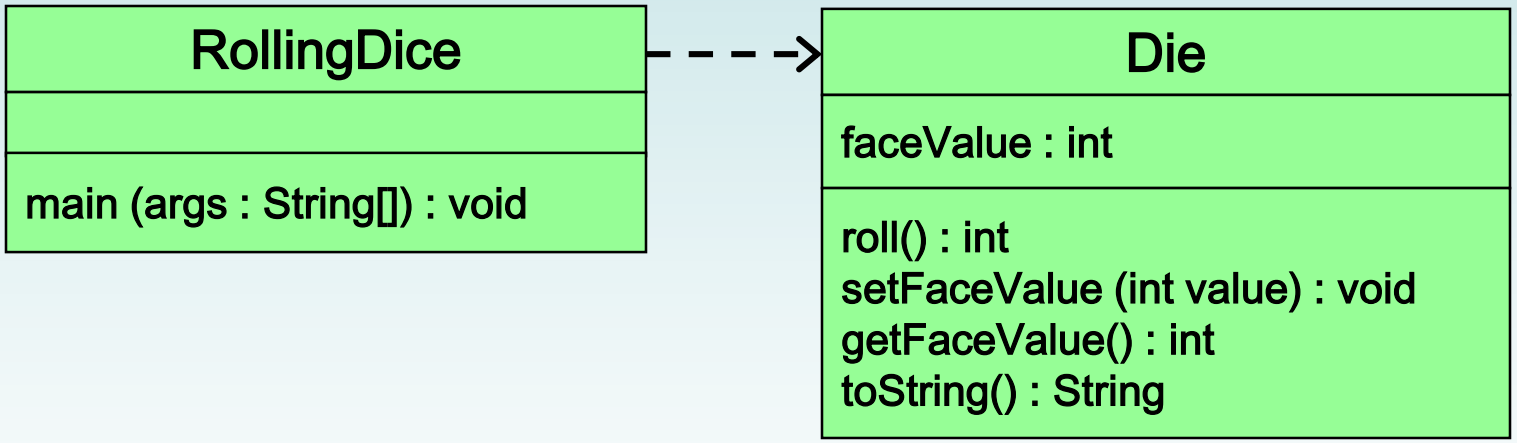
UML Diagrams

- UML stands for the *Unified Modeling Language*
- *UML diagrams* show relationships among classes and objects
- A UML *class diagram* consists of one or more classes, each with sections for the class name, attributes (data), and operations (methods)
- Lines between classes represent *associations*
- A dotted arrow shows that one class *uses* the other (calls its methods)

Copyright © 2014 Pearson Education, Inc.

UML Class Diagrams

- A UML class diagram for the `RollingDice` program:



Quick Check

What is the relationship between a class and an object?

Copyright © 2014 Pearson Education, Inc.

Quick Check

What is the relationship between a class and an object?

A class is the definition/pattern/blueprint of an object. It defines the data that will be managed by an object but doesn't reserve memory space for it. Multiple objects can be created from a class, and each object has its own copy of the instance data.

Copyright © 2014 Pearson Education, Inc.

Quick Check

Where is instance data declared?

What is the scope of instance data?

What is local data?

Copyright © 2014 Pearson Education, Inc.

Quick Check

Where is instance data declared?

At the class level.

What is the scope of instance data?

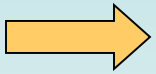
It can be referenced in any method of the class.

What is local data?

Local data is declared within a method, and is only accessible in that method.

Copyright © 2014 Pearson Education, Inc.

Outline



Anatomy of a Class

Encapsulation

Anatomy of a Method

Graphical Objects

Graphical User Interfaces

Buttons and Text Fields

Copyright © 2014 Pearson Education, Inc.

Encapsulation

- We can take one of two views of an object:
  - internal - the details of the variables and methods of the class that defines it
  - external - the services that an object provides and how the object interacts with the rest of the system
- From the external view, an object is an *encapsulated* entity, providing a set of specific services
- These services define the *interface* to the object

Copyright © 2014 Pearson Education, Inc.

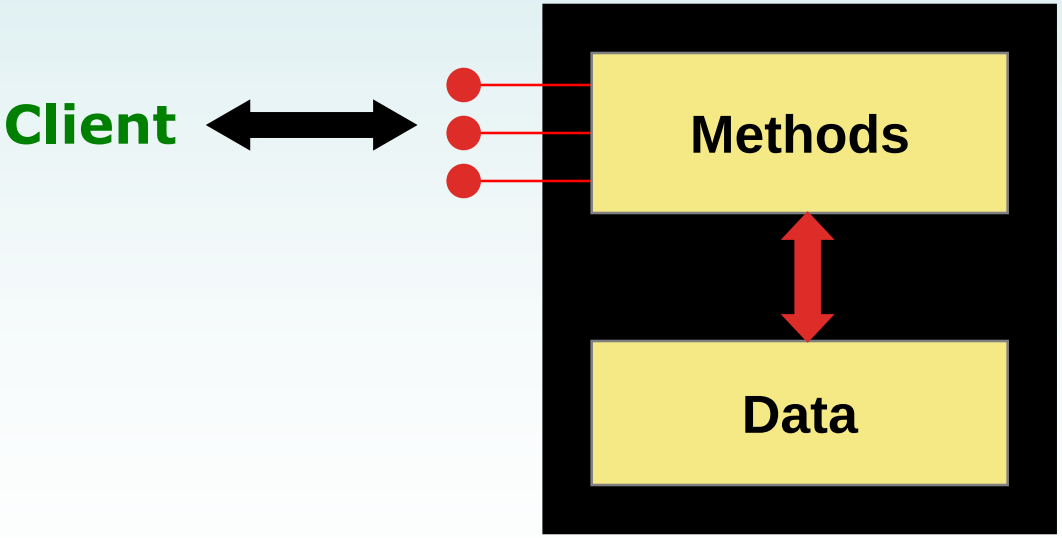
Encapsulation

- One object (called the *client*) may use another object for the services it provides
- The client of an object may request its services (call its methods), but it should not have to be aware of how those services are accomplished
- Any changes to the object's state (its variables) should be made by that object's methods
- We should make it difficult, if not impossible, for a client to access an object’s variables directly
- That is, an object should be *self-governing*

Copyright © 2014 Pearson Education, Inc.

Encapsulation

- An encapsulated object can be thought of as a *black box* -- its inner workings are hidden from the client
- The client invokes the interface methods and they manage the instance data



Copyright © 2014 Pearson Education, Inc.

Visibility Modifiers

- In Java, we accomplish encapsulation through the appropriate use of *visibility modifiers*
- A *modifier* is a Java reserved word that specifies particular characteristics of a method or data
- We've used the `final` modifier to define constants
- Java has three visibility modifiers: `public`, `protected`, and `private`
- The `protected` modifier involves inheritance, which we will discuss later

Copyright © 2014 Pearson Education, Inc.

Visibility Modifiers

- Members of a class that are declared with *public visibility* can be referenced anywhere
- Members of a class that are declared with *private visibility* can be referenced only within that class
- Members declared without a visibility modifier have *default visibility* and can be referenced by any class in the same package
- An overview of all Java modifiers is presented in Appendix E

Copyright © 2014 Pearson Education, Inc.

Visibility Modifiers

- Public variables violate encapsulation because they allow the client to modify the values directly
- Therefore instance variables should not be declared with public visibility
- It is acceptable to give a constant public visibility, which allows it to be used outside of the class
- Public constants do not violate encapsulation because, although the client can access it, its value cannot be changed

Copyright © 2014 Pearson Education, Inc.

Visibility Modifiers

- Methods that provide the object's services are declared with public visibility so that they can be invoked by clients
- Public methods are also called *service methods*
- A method created simply to assist a service method is called a *support method*
- Since a support method is not intended to be called by a client, it should not be declared with public visibility

Copyright © 2014 Pearson Education, Inc.

Visibility Modifiers

	`public`	`private`
Variables	**Violate encapsulation**	**Enforce encapsulation**
Methods	**Provide services to clients**	**Support other methods in the class**

Copyright © 2014 Pearson Education, Inc.

Accessors and Mutators

- Because instance data is private, a class usually provides services to access and modify data values
- An *accessor method* returns the current value of a variable
- A *mutator method* changes the value of a variable
- The names of accessor and mutator methods take the form `getX` and `setX`, respectively, where `X` is the name of the value
- They are sometimes called “getters” and “setters”

Copyright © 2014 Pearson Education, Inc.

Mutator Restrictions

- The use of mutators gives the class designer the ability to restrict a client's options to modify an object's state
- A mutator is often designed so that the values of variables can be set only within particular limits
- For example, the `setFaceValue` mutator of the `Die` class should restrict the value to the valid range (1 to `MAX`)
- We'll see in Chapter 5 how such restrictions can be implemented

Copyright © 2014 Pearson Education, Inc.

Quick Check

Why was the `faceValue` variable declared as `private` in the `Die` class?

Why is it ok to declare `MAX` as `public` in the `Die` class?

Copyright © 2014 Pearson Education, Inc.

Quick Check

Why was the `faceValue` variable declared as `private` in the `Die` class?

By making it private, each `Die` object controls its own data and allows it to be modified only by the well-defined operations it provides.

Why is it ok to declare `MAX` as `public` in the `Die` class?

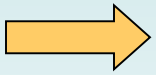
`MAX` is a constant. Its value cannot be changed. Therefore, there is no violation of encapsulation.

Copyright © 2014 Pearson Education, Inc.

Outline

Anatomy of a Class

Encapsulation



Anatomy of a Method

Graphical Objects

Graphical User Interfaces

Buttons and Text Fields

Copyright © 2014 Pearson Education, Inc.

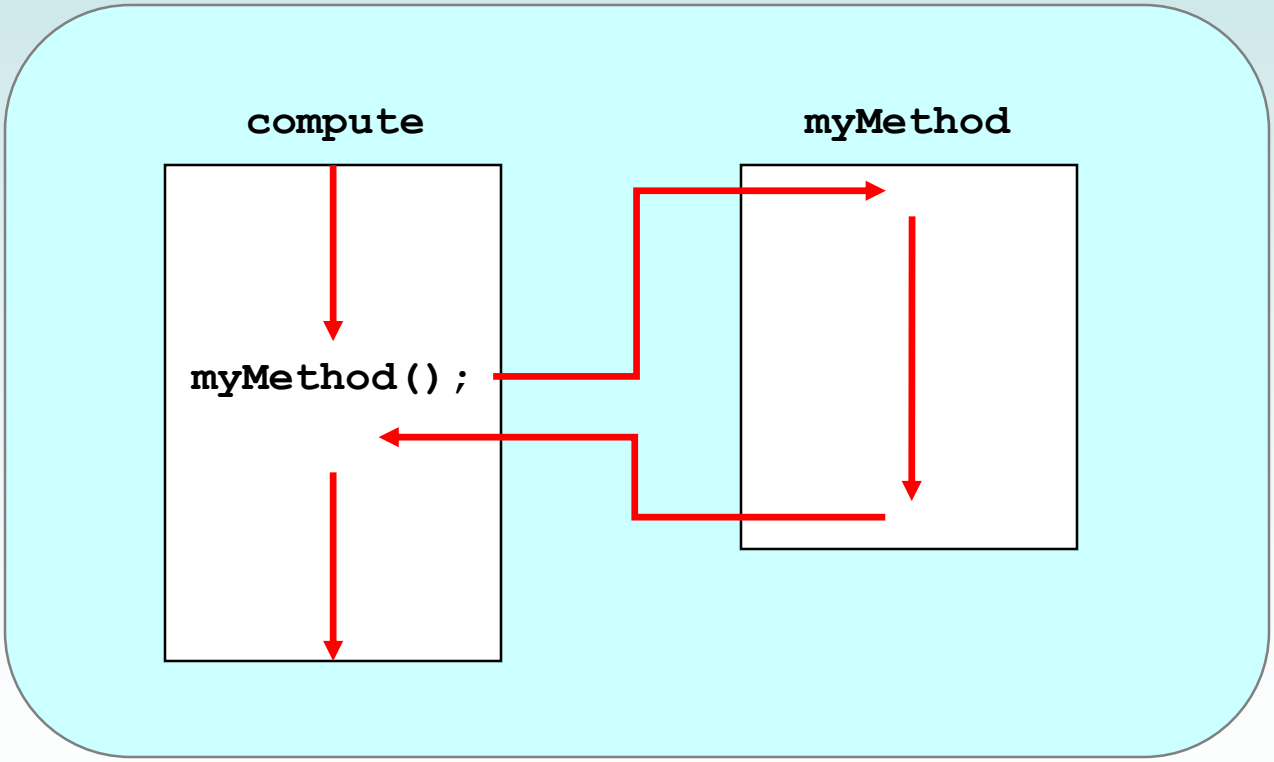
Method Declarations

- Let's now examine methods in more detail
- A *method declaration* specifies the code that will be executed when the method is invoked (called)
- When a method is invoked, the flow of control jumps to the method and executes its code
- When complete, the flow returns to the place where the method was called and continues
- The invocation may or may not return a value, depending on how the method is defined

Copyright © 2014 Pearson Education, Inc.

Method Control Flow

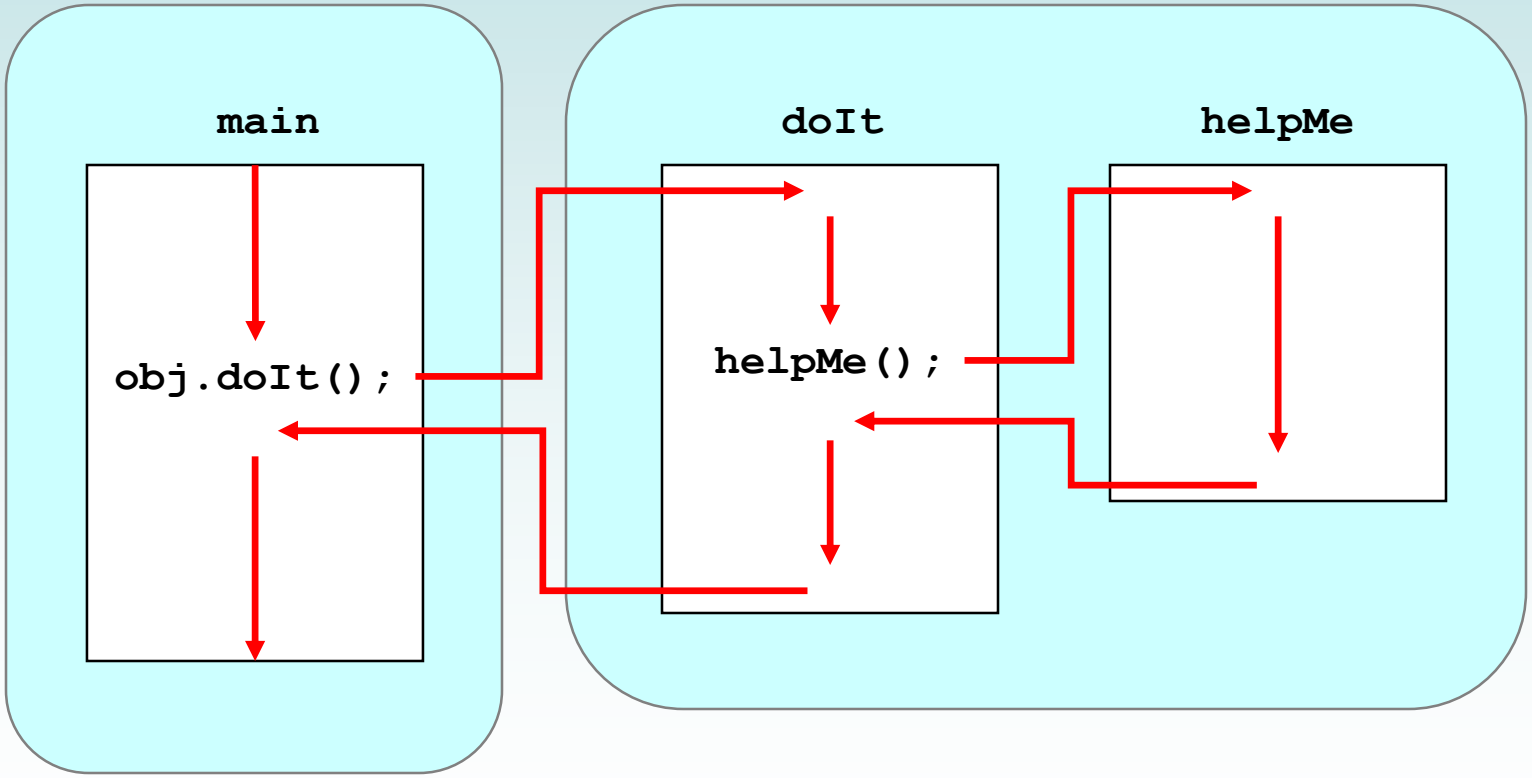
- If the called method is in the same class, only the method name is needed



Copyright © 2014 Pearson Education, Inc.

Method Control Flow

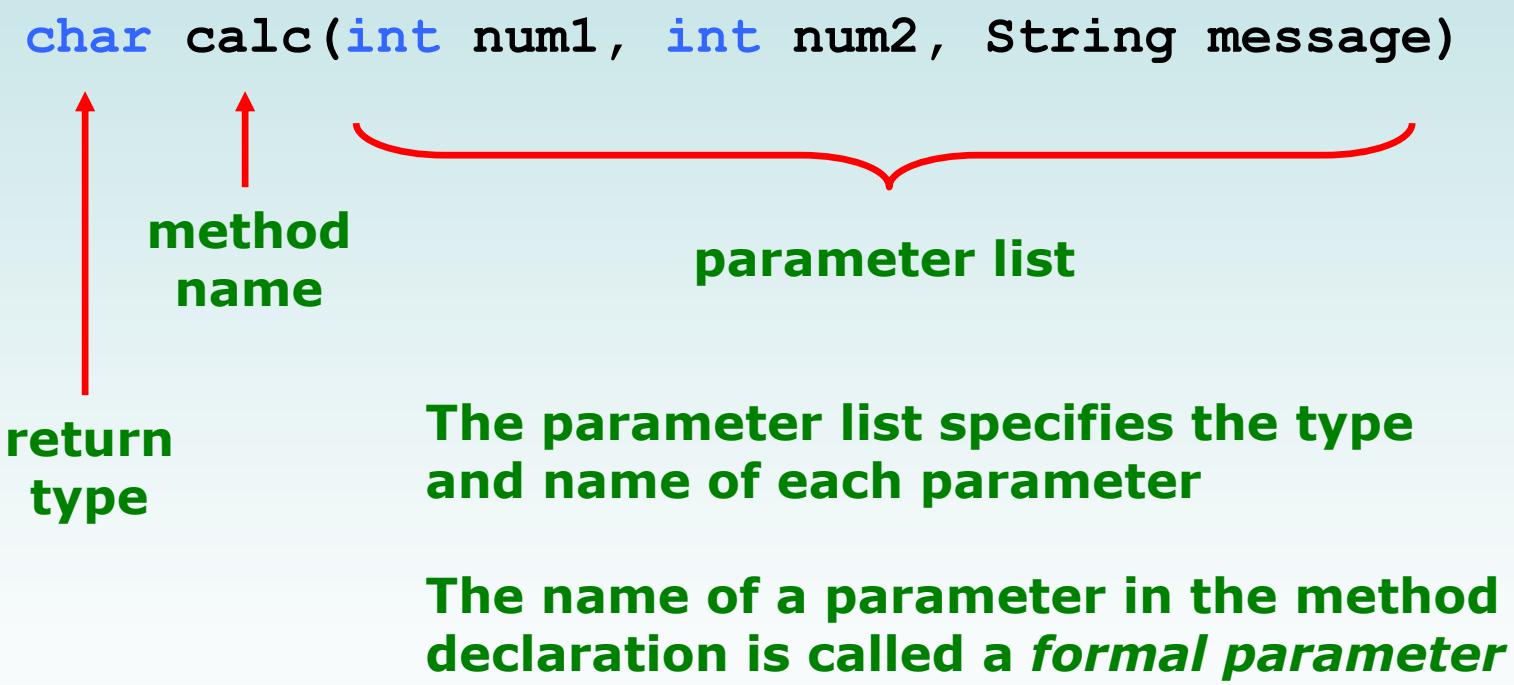
- The called method is often part of another class or object



Copyright © 2014 Pearson Education, Inc.

Method Header

- A method declaration begins with a *method header*



The parameter list specifies the type and name of each parameter

The name of a parameter in the method declaration is called a *formal parameter*

Copyright © 2014 Pearson Education, Inc.

Method Body

- The method header is followed by the *method body*

```
char calc(int num1, int num2, String message)
{
    int sum = num1 + num2;
    char result = message.charAt(sum);

    return result;
}
```

The return expression must be consistent with the return type

`sum` **and** `result` **are local data**

They are created each time the method is called, and are destroyed when it finishes executing

Copyright © 2014 Pearson Education, Inc.

The return Statement

- The *return type* of a method indicates the type of value that the method sends back to the calling location
- A method that does not return a value has a `void` return type
- A *return statement* specifies the value that will be returned

```
return expression;
```

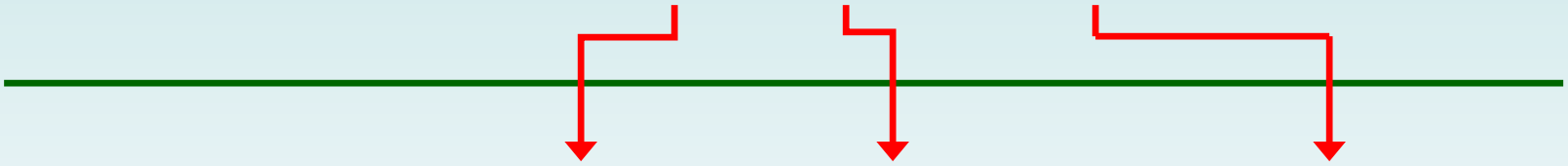
- Its expression must conform to the return type

Copyright © 2014 Pearson Education, Inc.

Parameters

- When a method is called, the *actual parameters* in the invocation are copied into the *formal parameters* in the method header

```
ch = obj.calc(25, count, "Hello");
```



```
char calc(int num1, int num2, String message)
{
    int sum = num1 + num2;
    char result = message.charAt(sum);

    return result;
}
```

Copyright © 2014 Pearson Education, Inc.

Local Data

- As we've seen, local variables can be declared inside a method
- The formal parameters of a method create *automatic local variables* when the method is invoked
- When the method finishes, all local variables are destroyed (including the formal parameters)
- Keep in mind that instance variables, declared at the class level, exists as long as the object exists

Copyright © 2014 Pearson Education, Inc.

Bank Account Example

- Let's look at another example that demonstrates the implementation details of classes and methods
- We'll represent a bank account by a class named `Account`
- It's state can include the account number, the current balance, and the name of the owner
- An account's behaviors (or services) include deposits and withdrawals, and adding interest

Copyright © 2014 Pearson Education, Inc.

Driver Programs

- A *driver program* drives the use of other, more interesting parts of a program
- Driver programs are often used to test other parts of the software
- The `Transactions` class contains a `main` method that drives the use of the `Account` class, exercising its services
- See `Transactions.java`
- See `Account.java`

Copyright © 2014 Pearson Education, Inc.

```
//********************************************************************
//  Transactions.java       Author: Lewis/Loftus
//
//  Demonstrates the creation and use of multiple Account objects.
//********************************************************************

public class Transactions
{
   //-----------------------------------------------------------------
   //  Creates some bank accounts and requests various services.
   //-----------------------------------------------------------------
   public static void main(String[] args)
   {
      Account acct1 = new Account("Ted Murphy", 72354, 102.56);
      Account acct2 = new Account("Jane Smith", 69713, 40.00);
      Account acct3 = new Account("Edward Demsey", 93757, 759.32);

      acct1.deposit(25.85);

      double smithBalance = acct2.deposit(500.00);
      System.out.println("Smith balance after deposit: " +
                         smithBalance);
```

continue

Copyright © 2014 Pearson Education, Inc.

continue

```
      System.out.println("Smith balance after withdrawal: " +
                         acct2.withdraw (430.75, 1.50));

      acct1.addInterest();
      acct2.addInterest();
      acct3.addInterest();

      System.out.println();
      System.out.println(acct1);
      System.out.println(acct2);
      System.out.println(acct3);
   }
}
```

Copyright © 2014 Pearson Education, Inc.

continue

```
        System.                                                    +


        acct1.a
        acct2.a
        acct3.a

        System.out.println();
        System.out.println(acct1);
        System.out.println(acct2);
        System.out.println(acct3);
    }
}
```

Output

```
Smith balance after deposit: 540.0
Smith balance after withdrawal: 107.55

72354   Ted Murphy      $132.90
69713   Jane Smith      $111.52
93757   Edward Demsey   $785.90
```

Copyright © 2014 Pearson Education, Inc.

```
//********************************************************************
//  Account.java       Author: Lewis/Loftus
//
//  Represents a bank account with basic services such as deposit
//  and withdraw.
//********************************************************************

import java.text.NumberFormat;

public class Account
{
   private final double RATE = 0.035;  // interest rate of 3.5%

   private long acctNumber;
   private double balance;
   private String name;

   //-----------------------------------------------------------------
   //  Sets up the account by defining its owner, account number,
   //  and initial balance.
   //-----------------------------------------------------------------
   public Account(String owner, long account, double initial)
   {
      name = owner;
      acctNumber = account;
      balance = initial;
   }
```

continue

, Inc.

continue

```
   //-----------------------------------------------------------------
   //  Deposits the specified amount into the account. Returns the
   //  new balance.
   //-----------------------------------------------------------------
   public double deposit(double amount)
   {
      balance = balance + amount;
      return balance;
   }

   //-----------------------------------------------------------------
   //  Withdraws the specified amount from the account and applies
   //  the fee. Returns the new balance.
   //-----------------------------------------------------------------
   public double withdraw(double amount, double fee)
   {
      balance = balance - amount - fee;
      return balance;
   }
```

continue

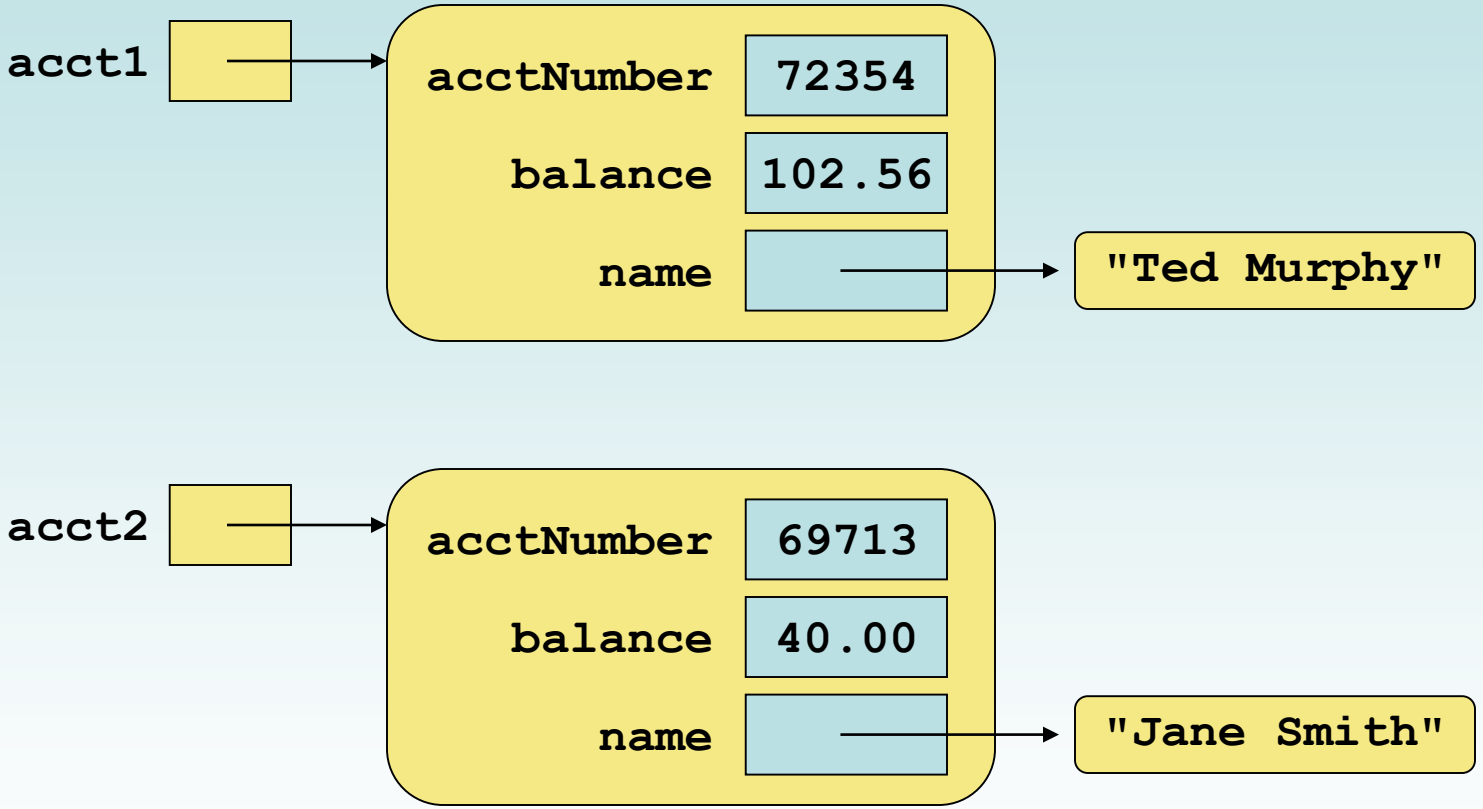
Copyright © 2014 Pearson Education, Inc.

continue

```
    //-----------------------------------------------------------------
    //  Adds interest to the account and returns the new balance.
    //-----------------------------------------------------------------
    public double addInterest()
    {
       balance += (balance * RATE);
       return balance;
    }

    //-----------------------------------------------------------------
    //  Returns the current balance of the account.
    //-----------------------------------------------------------------
    public double getBalance()
    {
       return balance;
    }

    //-----------------------------------------------------------------
    //  Returns a one-line description of the account as a string.
    //-----------------------------------------------------------------
    public String toString()
    {
       NumberFormat fmt = NumberFormat.getCurrencyInstance();
       return (acctNumber + "\t" + name + "\t" + fmt.format(balance));
    }
}
```

Copyright © 2014 Pearson Education, Inc.

Bank Account Example



Copyright © 2014 Pearson Education, Inc.

Bank Account Example

- There are some improvements that can be made to the `Account` class
- Formal getters and setters could have been defined for all data
- The design of some methods could also be more robust, such as verifying that the `amount` parameter to the `withdraw` method is positive

Copyright © 2014 Pearson Education, Inc.

Constructors Revisited

- Note that a constructor has no return type specified in the method header, not even `void`
- A common error is to put a return type on a constructor, which makes it a “regular” method that happens to have the same name as the class
- The programmer does not have to define a constructor for a class
- Each class has a *default constructor* that accepts no parameters

Copyright © 2014 Pearson Education, Inc.

Quick Check

How do we express which `Account` object's balance is updated when a deposit is made?

Copyright © 2014 Pearson Education, Inc.

Quick Check

How do we express which `Account` object's balance is updated when a deposit is made?

Each account is referenced by an object reference variable:

```
Account myAcct = new Account(…);
```

and when a method is called, you call it through a particular object:

```
myAcct.deposit(50);
```

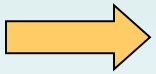
Copyright © 2014 Pearson Education, Inc.

Outline

Anatomy of a Class

Encapsulation

Anatomy of a Method



Graphical Objects

Graphical User Interfaces

Buttons and Text Fields

Copyright © 2014 Pearson Education, Inc.

Graphical Objects

- Some objects contain information that determines how the object should be represented visually
- Most GUI components are graphical objects
- We can have some effect on how components get drawn
- We did this in Chapter 2 when we defined the `paint` method of an applet
- Let's look at some other examples of graphical objects

Copyright © 2014 Pearson Education, Inc.

Smiling Face Example

- The `SmilingFace` program draws a face by defining the `paintComponent` method of a panel
- See `SmilingFace.java`
- See `SmilingFacePanel.java`
- The `main` method of the `SmilingFace` class instantiates a `SmilingFacePanel` and displays it
- The `SmilingFacePanel` class is derived from the `JPanel` class using inheritance

Copyright © 2014 Pearson Education, Inc.

```
//********************************************************************
//  SmilingFace.java       Author: Lewis/Loftus
//
//  Demonstrates the use of a separate panel class.
//********************************************************************

import javax.swing.JFrame;

public class SmilingFace
{
   //-----------------------------------------------------------------
   //  Creates the main frame of the program.
   //-----------------------------------------------------------------
   public static void main(String[] args)
   {
      JFrame frame = new JFrame("Smiling Face");
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

      SmilingFacePanel panel = new SmilingFacePanel();

      frame.getContentPane().add(panel);

      frame.pack();
      frame.setVisible(true);
   }
}
```

Copyright © 2014 Pearson Education, Inc.

```
//************
//  SmilingFac
//
//  Demonstrat
//************

import javax.s

public class S
{
   //--------
   //  Creates
   //--------
   public stat
   {
      JFrame frame = new JFrame("Smiling Face");
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

      SmilingFacePanel panel = new SmilingFacePanel();

      frame.getContentPane().add(panel);

      frame.pack();
      frame.setVisible(true);
   }
}
```

Copyright © 2014 Pearson Education, Inc.

```
//********************************************************************
//  SmilingFacePanel.java       Author: Lewis/Loftus
//
//  Demonstrates the use of a separate panel class.
//********************************************************************

import javax.swing.JPanel;
import java.awt.*;

public class SmilingFacePanel extends JPanel
{
   private final int BASEX = 120, BASEY = 60; // base point for head

   //-----------------------------------------------------------------
   //  Constructor: Sets up the main characteristics of this panel.
   //-----------------------------------------------------------------
   public SmilingFacePanel()
   {
      setBackground(Color.blue);
      setPreferredSize(new Dimension(320, 200));
      setFont(new Font("Arial", Font.BOLD, 16));
   }
```

continue

Copyright © 2014 Pearson Education, Inc.

continue

```
    //-----------------------------------------------------------------
    //  Draws a face.
    //-----------------------------------------------------------------
    public void paintComponent(Graphics page)
    {
       super.paintComponent(page);

       page.setColor(Color.yellow);
       page.fillOval(BASEX, BASEY, 80, 80);  // head
       page.fillOval(BASEX-5, BASEY+20, 90, 40);  // ears

       page.setColor(Color.black);
       page.drawOval(BASEX+20, BASEY+30, 15, 7);  // eyes
       page.drawOval(BASEX+45, BASEY+30, 15, 7);

       page.fillOval(BASEX+25, BASEY+31, 5, 5);   // pupils
       page.fillOval(BASEX+50, BASEY+31, 5, 5);

       page.drawArc(BASEX+20, BASEY+25, 15, 7, 0, 180);  // eyebrows
       page.drawArc(BASEX+45, BASEY+25, 15, 7, 0, 180);

       page.drawArc(BASEX+35, BASEY+40, 15, 10, 180, 180);  // nose
       page.drawArc(BASEX+20, BASEY+50, 40, 15, 180, 180);  // mouth
```

continue

Copyright © 2014 Pearson Education, Inc.

continue

```
        page.setColor(Color.white);
        page.drawString("Always remember that you are unique!",
                        BASEX-105, BASEY-15);
        page.drawString("Just like everyone else.", BASEX-45, BASEY+105);
    }
}
```

Copyright © 2014 Pearson Education, Inc.

Smiling Face Example

- Every Swing component has a `paintComponent` method
- The `paintComponent` method accepts a `Graphics` object that represents the graphics context for the panel
- We define the `paintComponent` method to draw the face with appropriate calls to the `Graphics` methods
- Note the difference between drawing on a panel and adding other GUI components to a panel

Copyright © 2014 Pearson Education, Inc.

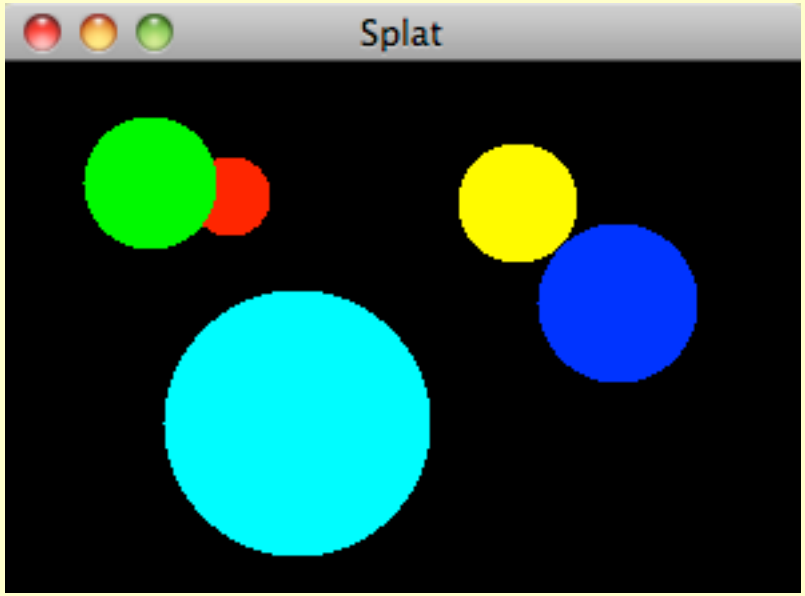
Splat Example

- The `Splat` example is structured a bit differently
- It draws a set of colored circles on a panel, but each circle is represented as a separate object that maintains its own graphical information
- The `paintComponent` method of the panel "asks" each circle to draw itself
- See `Splat.java`
- See `SplatPanel.java`
- See `Circle.java`

Copyright © 2014 Pearson Education, Inc.

```
//********************************************************************
//  Splat.java       Author: Lewis/Loftus
//
//  Demonstrates the use of graphical objects.
//********************************************************************

import javax.swing.*;
import java.awt.*;

public class Splat
{
   //-----------------------------------------------------------------
   //  Presents a collection of circles.
   //-----------------------------------------------------------------
   public static void main(String[] args)
   {
      JFrame frame = new JFrame("Splat");
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

      frame.getContentPane().add(new SplatPanel());

      frame.pack();
      frame.setVisible(true);
   }
}
```

Copyright © 2014 Pearson Education, Inc.




```
//*************
//  Splat.java
//
//  Demonstrate
//*************

import javax.sw
import java.awt

public class Sp
{
   //----------
   //  Presents
   //----------
   public static void main(String[] args)
   {
      JFrame frame = new JFrame("Splat");
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

      frame.getContentPane().add(new SplatPanel());

      frame.pack();
      frame.setVisible(true);
   }
}
```

Copyright © 2014 Pearson Education, Inc.

```
//********************************************************************
//  SplatPanel.java       Author: Lewis/Loftus
//
//  Demonstrates the use of graphical objects.
//********************************************************************

import javax.swing.*;
import java.awt.*;

public class SplatPanel extends JPanel
{
   private Circle circle1, circle2, circle3, circle4, circle5;

   //-----------------------------------------------------------------
   //  Constructor: Creates five Circle objects.
   //-----------------------------------------------------------------
   public SplatPanel()
   {
      circle1 = new Circle(30, Color.red, 70, 35);
      circle2 = new Circle(50, Color.green, 30, 20);
      circle3 = new Circle(100, Color.cyan, 60, 85);
      circle4 = new Circle(45, Color.yellow, 170, 30);
      circle5 = new Circle(60, Color.blue, 200, 60);

      setPreferredSize(new Dimension(300, 200));
      setBackground(Color.black);
   }
```

continue

, Inc.

continue

```
    //-----------------------------------------------------------------
    //  Draws this panel by requesting that each circle draw itself.
    //-----------------------------------------------------------------
    public void paintComponent(Graphics page)
    {
        super.paintComponent(page);

        circle1.draw(page);
        circle2.draw(page);
        circle3.draw(page);
        circle4.draw(page);
        circle5.draw(page);
    }
}
```

Copyright © 2014 Pearson Education, Inc.

```
//********************************************************************
//  Circle.java       Author: Lewis/Loftus
//
//  Represents a circle with a particular position, size, and color.
//********************************************************************

import java.awt.*;

public class Circle
{
   private int diameter, x, y;
   private Color color;

   //-----------------------------------------------------------------
   //  Constructor: Sets up this circle with the specified values.
   //-----------------------------------------------------------------
   public Circle(int size, Color shade, int upperX, int upperY)
   {
      diameter = size;
      color = shade;
      x = upperX;
      y = upperY;
   }
```

continue

Copyright © 2014 Pearson Education, Inc.

continue

```
    //-----------------------------------------------------------------
    //  Draws this circle in the specified graphics context.
    //-----------------------------------------------------------------
    public void draw(Graphics page)
    {
       page.setColor(color);
       page.fillOval(x, y, diameter, diameter);
    }

    //-----------------------------------------------------------------
    //  Diameter mutator.
    //-----------------------------------------------------------------
    public void setDiameter(int size)
    {
       diameter = size;
    }

    //-----------------------------------------------------------------
    //  Color mutator.
    //-----------------------------------------------------------------
    public void setColor(Color shade)
    {
       color = shade;
    }
```

continue

Copyright © 2014 Pearson Education, Inc.

continue

```
    //-----------------------------------------------------------------
    //  X mutator.
    //-----------------------------------------------------------------
    public void setX(int upperX)
    {
       x = upperX;
    }

    //-----------------------------------------------------------------
    //  Y mutator.
    //-----------------------------------------------------------------
    public void setY(int upperY)
    {
       y = upperY;
    }

    //-----------------------------------------------------------------
    //  Diameter accessor.
    //-----------------------------------------------------------------
    public int getDiameter()
    {
       return diameter;
    }
```

continue

Copyright © 2014 Pearson Education, Inc.

continue

```
    //-----------------------------------------------------------------
    //  Color accessor.
    //-----------------------------------------------------------------
    public Color getColor()
    {
       return color;
    }

    //-----------------------------------------------------------------
    //  X accessor.
    //-----------------------------------------------------------------
    public int getX()
    {
       return x;
    }

    //-----------------------------------------------------------------
    //  Y accessor.
    //-----------------------------------------------------------------
    public int getY()
    {
       return y;
    }
}
```

Copyright © 2014 Pearson Education, Inc.

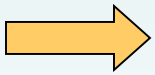
Outline

Anatomy of a Class

Encapsulation

Anatomy of a Method

Graphical Objects



Graphical User Interfaces

Buttons and Text Fields

Copyright © 2014 Pearson Education, Inc.

Graphical User Interfaces

- A Graphical User Interface (GUI) in Java is created with at least three kinds of objects:
  - components, events, and listeners
- We've previously discussed *components*, which are objects that represent screen elements:
  - labels, buttons, text fields, menus, etc.
- Some components are *containers* that hold and organize other components:
  - frames, panels, applets, dialog boxes

Copyright © 2014 Pearson Education, Inc.

Events

- An *event* is an object that represents some activity to which we may want to respond
- For example, we may want our program to perform some action when the following occurs:
  - the mouse is moved
  - the mouse is dragged
  - a mouse button is clicked
  - a graphical button is pressed
  - a keyboard key is pressed
  - a timer expires

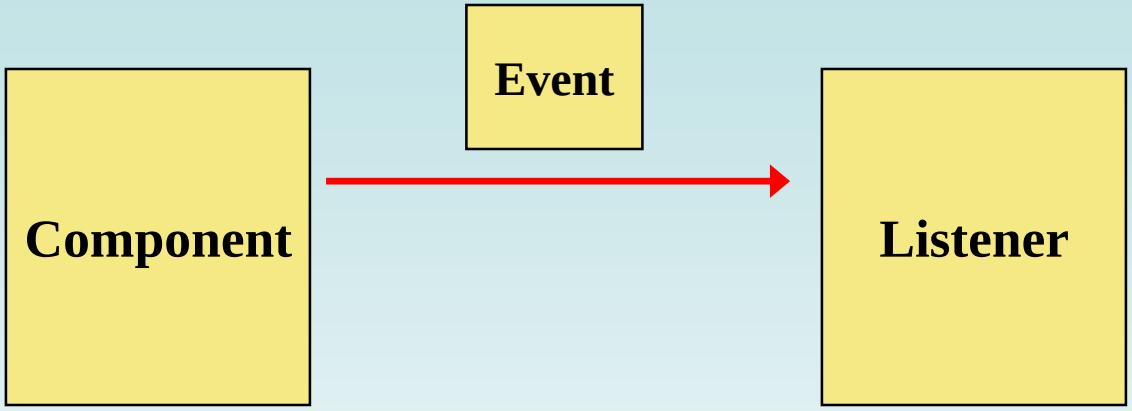
Copyright © 2014 Pearson Education, Inc.

Events and Listeners

- The Java API contains several classes that represent typical events
- Components, such as a graphical button, generate (or fire) an event when it occurs
- We set up a *listener* object to respond to an event when it occurs
- We can design listener objects to take whatever actions are appropriate when an event occurs

Copyright © 2014 Pearson Education, Inc.

Events and Listeners



A component object generates an event

A corresponding listener object is designed to respond to the event

When the event occurs, the component calls the appropriate method of the listener, passing an object that describes the event

Copyright © 2014 Pearson Education, Inc.

GUI Development

- To create a Java program that uses a GUI we must:
  - instantiate and set up the necessary components
  - implement listener classes for any events we care about
  - establish the relationship between listeners and the components that generate the corresponding events
- Let's now explore some new components and see how this all comes together

Copyright © 2014 Pearson Education, Inc.

Outline

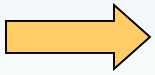
Anatomy of a Class

Encapsulation

Anatomy of a Method

Graphical Objects

Graphical User Interfaces



Buttons and Text Fields

Copyright © 2014 Pearson Education, Inc.

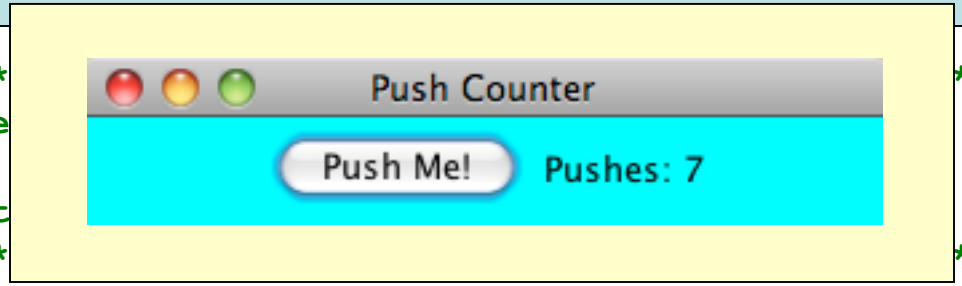
Buttons

- A *push button* is defined by the `JButton` class
- It generates an *action event*
- The `PushCounter` example displays a push button that increments a counter each time it is pushed
- See `PushCounter.java`
- See `PushCounterPanel.java`

Copyright © 2014 Pearson Education, Inc.

```
//********************************************************************
//  PushCounter.java       Authors: Lewis/Loftus
//
//  Demonstrates a graphical user interface and an event listener.
//********************************************************************

import javax.swing.JFrame;

public class PushCounter
{
   //-----------------------------------------------------------------
   //  Creates the main program frame.
   //-----------------------------------------------------------------
   public static void main(String[] args)
   {
      JFrame frame = new JFrame("Push Counter");
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

      frame.getContentPane().add(new PushCounterPanel());

      frame.pack();
      frame.setVisible(true);
   }
}
```

Copyright © 2014 Pearson Education, Inc.




```
//*************                                              **************
//  PushCounte
//
//  Demonstrat                                               listener.
//*************                                              **************

import javax.swing.JFrame;

public class PushCounter
{
   //-----------------------------------------------------------------
   //  Creates the main program frame.
   //-----------------------------------------------------------------
   public static void main(String[] args)
   {
      JFrame frame = new JFrame("Push Counter");
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

      frame.getContentPane().add(new PushCounterPanel());

      frame.pack();
      frame.setVisible(true);
   }
}
```

Copyright © 2014 Pearson Education, Inc.

```
//********************************************************************
//  PushCounterPanel.java       Authors: Lewis/Loftus
//
//  Demonstrates a graphical user interface and an event listener.
//********************************************************************

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class PushCounterPanel extends JPanel
{
   private int count;
   private JButton push;
   private JLabel label;

   //-----------------------------------------------------------------
   //  Constructor: Sets up the GUI.
   //-----------------------------------------------------------------
   public PushCounterPanel()
   {
      count = 0;

      push = new JButton("Push Me!");
      push.addActionListener(new ButtonListener());
```

continue

Copyright © 2014 Pearson Education, Inc.

continue

```
        label = new JLabel("Pushes: " + count);

        add(push);
        add(label);

        setPreferredSize(new Dimension(300, 40));
        setBackground(Color.cyan);
    }

    //*****************************************************************
    //  Represents a listener for button push (action) events.
    //*****************************************************************
    private class ButtonListener implements ActionListener
    {
        //--------------------------------------------------------------
        //  Updates the counter and label when the button is pushed.
        //--------------------------------------------------------------
        public void actionPerformed(ActionEvent event)
        {
            count++;
            label.setText("Pushes: " + count);
        }
    }
}
```

Copyright © 2014 Pearson Education, Inc.

Push Counter Example

- The components of the GUI are the button, a label to display the counter, a panel to organize the components, and the main frame
- The `PushCounterPanel` class represents the panel used to display the button and label
- The `PushCounterPanel` class is derived from `JPanel` using inheritance
- The constructor of `PushCounterPanel` sets up the elements of the GUI and initializes the counter to zero

Copyright © 2014 Pearson Education, Inc.

Push Counter Example

- The `ButtonListener` class is the listener for the action event generated by the button
- It is implemented as an *inner class*, which means it is defined within the body of another class
- That facilitates the communication between the listener and the GUI components
- Inner classes should only be used in situations where there is an intimate relationship between the two classes and the inner class is not needed in any other context

Copyright © 2014 Pearson Education, Inc.

Push Counter Example

- Listener classes are written by implementing a *listener interface*
- The `ButtonListener` class implements the `ActionListener` interface
- An interface is a list of methods that the implementing class must define
- The only method in the `ActionListener` interface is the `actionPerformed` method
- The Java API contains interfaces for many types of events
- We discuss interfaces in more detail in Chapter 6

Copyright © 2014 Pearson Education, Inc.

Push Counter Example

- The `PushCounterPanel` constructor:
  - instantiates the `ButtonListener` object
  - establishes the relationship between the button and the listener by the call to `addActionListener`
- When the user presses the button, the button component creates an `ActionEvent` object and calls the `actionPerformed` method of the listener
- The `actionPerformed` method increments the counter and resets the text of the label

Copyright © 2014 Pearson Education, Inc.

Quick Check

Which object in the Push Counter example generated the event?

What did it do then?

Copyright © 2014 Pearson Education, Inc.

Quick Check

Which object in the Push Counter example generated the event?

The button component generated the event.

What did it do then?

It called the `actionPerformed` method of the listener object that had been registered with it.

Copyright © 2014 Pearson Education, Inc.

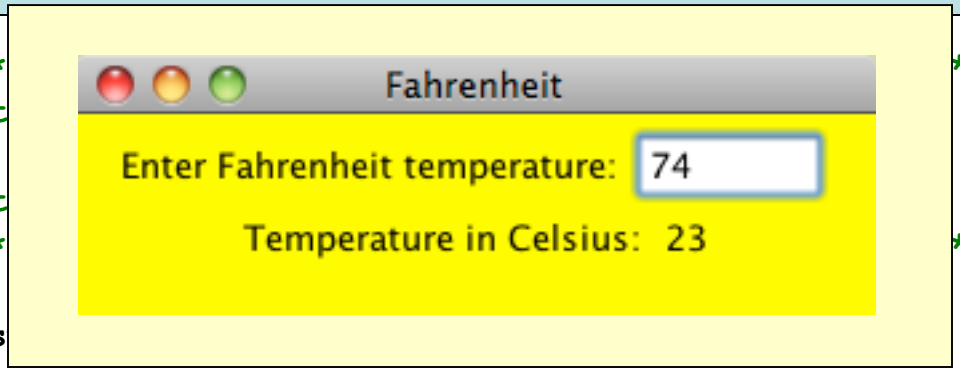
Text Fields

- Let's look at another GUI example that uses another type of component
- A *text field* allows the user to enter one line of input
- If the cursor is in the text field, the text field object generates an action event when the enter key is pressed
- See `Fahrenheit.java`
- See `FahrenheitPanel.java`

Copyright © 2014 Pearson Education, Inc.

```
//********************************************************************
//  Fahrenheit.java       Author: Lewis/Loftus
//
//  Demonstrates the use of text fields.
//********************************************************************

import javax.swing.JFrame;

public class Fahrenheit
{
   //-----------------------------------------------------------------
   //  Creates and displays the temperature converter GUI.
   //-----------------------------------------------------------------
   public static void main(String[] args)
   {
      JFrame frame = new JFrame("Fahrenheit");
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

      FahrenheitPanel panel = new FahrenheitPanel();

      frame.getContentPane().add(panel);
      frame.pack();
      frame.setVisible(true);
   }
}
```

Copyright © 2014 Pearson Education, Inc.




```
//************
//  Fahrenheit
//
//  Demonstrat
//************

import javax.s

public class Fahrenheit
{
   //-----------------------------------------------------------------
   //  Creates and displays the temperature converter GUI.
   //-----------------------------------------------------------------
   public static void main(String[] args)
   {
      JFrame frame = new JFrame("Fahrenheit");
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

      FahrenheitPanel panel = new FahrenheitPanel();

      frame.getContentPane().add(panel);
      frame.pack();
      frame.setVisible(true);
   }
}
```

Copyright © 2014 Pearson Education, Inc.

```
//********************************************************************
//  FahrenheitPanel.java       Author: Lewis/Loftus
//
//  Demonstrates the use of text fields.
//********************************************************************

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class FahrenheitPanel extends JPanel
{
   private JLabel inputLabel, outputLabel, resultLabel;
   private JTextField fahrenheit;

   //-----------------------------------------------------------------
   //  Constructor: Sets up the main GUI components.
   //-----------------------------------------------------------------
   public FahrenheitPanel()
   {
      inputLabel = new JLabel("Enter Fahrenheit temperature:");
      outputLabel = new JLabel("Temperature in Celsius: ");
      resultLabel = new JLabel("---");

      fahrenheit = new JTextField(5);
      fahrenheit.addActionListener(new TempListener());
```

continue

Copyright © 2014 Pearson Education, Inc.

continue

```
        add(inputLabel);
        add(fahrenheit);
        add(outputLabel);
        add(resultLabel);

        setPreferredSize(new Dimension(300, 75));
        setBackground(Color.yellow);
    }

    //*****************************************************************
    //  Represents an action listener for the temperature input field.
    //*****************************************************************
    private class TempListener implements ActionListener
    {
        //--------------------------------------------------------------
        //  Performs the conversion when the enter key is pressed in
        //  the text field.
        //--------------------------------------------------------------
        public void actionPerformed(ActionEvent event)
        {
            int fahrenheitTemp, celsiusTemp;

            String text = fahrenheit.getText();
```

continue

Copyright © 2014 Pearson Education, Inc.

continue

```
            fahrenheitTemp = Integer.parseInt(text);
            celsiusTemp = (fahrenheitTemp-32) * 5/9;

            resultLabel.setText(Integer.toString(celsiusTemp));
        }
    }
}
```

Copyright © 2014 Pearson Education, Inc.

Fahrenheit Example

- Like the `PushCounter` example, the GUI is set up in a separate panel class
- The `TempListener` inner class defines the listener for the action event generated by the text field
- The `FahrenheitPanel` constructor instantiates the listener and adds it to the text field
- When the user types a temperature and presses enter, the text field generates the action event and calls the `actionPerformed` method of the listener

Copyright © 2014 Pearson Education, Inc.

Summary

- Chapter 4 focused on:
  - class definitions
  - instance data
  - encapsulation and Java modifiers
  - method declaration and parameter passing
  - constructors
  - graphical objects
  - events and listeners
  - buttons and text fields

Copyright © 2014 Pearson Education, Inc.